(a) Given M_a, w: Does M_a halt on input w?

(b) Given M_b: Does M_b halt on input ε?

Given that (a) is not Turing-decidable, prove that (b) is not Turing-decidable.

That is, assume that you have an algorithm that solves problem (b). Prove that with an appropriate transformation of the input, you could use it to solve problem (a).

An *empty-stack* PDA accepts a string w if there exists some computation that starts in the start state, applies legal transitions at each step and terminates in a final state with all the input consumed, and an empty stack.

A *stack-oblivious* PDA accepts a string w if there exists some computation that starts in the start state, applies legal transitions at each step and terminates in a final state with all the input consumed, and it does not matter if there is still something in the stack or not.

Last year,
some students did this to convert from a stack-oblivious PDA to an empty-stack one:

For each final state f, and each symbol σ that is in the stack alphabet, add a rule:

State	Read	Pop	Next state	Push
f	ε	σ	f	ε

Why does this not work?

Counterexample: Start: s, Final {u, w}
$L = \{a^n b^m : m \leq n\} \cup \{a^n b^m c^p : n = m+p\}$

State	Read	Pop	Next state	Push
s	ε	ε	t	$
t	a	ε	t	x
t	ε	ε	u	ε
u	b	x	u	ε
u	ε	ε	v	ε
v	c	x	v	ε
v	ε	$	w	ε

Announcements

Assignment #5 has been posted.
Due Friday Aug. 3 at the beginning of class.
We will have a tutorial this week.

Final exam tutorial:

Monday Aug. 6, 10am, ECS 116.

If the building is locked, I will prop open the back door to ECS (the one that opens on to the campus).

Old finals are available from the course web page.

L= { "M" : there is some string w in {a,b}* such that M halts on input w}

Theorem: L is Turing-acceptable.

Proof: Example of dovetailing.

Proving Problems Undecidable

To prove problem Q is not decidable:

1. Select a problem P known to not be decidable.
2. Prove that if you can solve Q you can solve P.
3. This is a contradiction since P is not decidable and therefore Q is not decidable.

Known: H_1 = {"M": M halts on input "M"} is not Turing decidable.

Theorem 5.4.2 (p. 255): The following problems are not Turing-decidable:

(a) Given M, w: Does M halt on input w?

(b) Given M: Does M halt on input ε?

(c) Given M: Is there any string on which M halts?

(d) Given M: Does M halt on every string?

(e) Given two TM's M_1 and M_2: Do they halt on the same input strings?

The following languages are Turing-decidable:

(a) {"M" "w" : M halts on w after at most 700 steps}

(b) {"M" "w": M halts on w without using more than the first 100 tape squares}

For part (b): we only have to simulate M for a finite number of steps and in that time frame it will either halt, hang, use more than the first 100 tape squares, or it will never halt. How many steps must we run M for?

Theorem: Turing decidable languages are closed under complement.

Proof:

Let M be a TM which decides L.

It is easy to construct the machine schema for a TM which decides the complement of L.

Algorithm: Run M. Change Y to N and N to Y at end then position head appropriately.

Theorem: All Turing-decidable languages are Turing-acceptable.

Recall:

Decide means to halt with (h, #Y[#]) when w is in L and (h, #N[#]) when w is not in L.

Accept means that the TM halts on w when w is in L and hangs (head falls off left hand end of tape or there is an undefined transition) or computes forever when w is not in L.

Proof: Given a TM M_1 that decides L we can easily design a machine M_2 which accepts L.

Theorem: Turing-decidable languages are closed under Kleene star.

Example: w= abcd

Which factorizations of w must be considered?

w_1	w_2	w_3	w_4
a	b	c	d
a	b	cd	
a	bc	d	
a	bcd		
ab	c	d	
ab	cd		Kleene
abc	d		Star
abcd			Cases

```
public static void testW(int level, String w)
{
    int i, j, len;  String u, v;

    len= w.length( ); if (len == 0) return;

    for (i=1; i <= len; i++)
    {
        u= w.substring(0,i);
        for (j=0; j < level-1; j++)
             System.out.print("            ");
        System.out.println(
            "W" + level + " = " + u);
        v= w.substring(i, len);
        testW(level+1, v);
    }
}
```

Thought question: what would you do to determine if a string w is in $L_1 \cdot L_2$ if you have TM's which decide L_1 and L_2?

High level pseudo code is fine. It would not be fun to program this on a TM.

Summary: Closure

Operation	Turing-decidable	Turing-acceptable
Union	yes	yes
Concatenation	yes	yes
Kleene star	yes	yes
Complement	yes	no
Intersection	yes	yes

Suppose I construct a TM M_f which:

1. Preserves its input u.

2. Simulates a machine M_b on input ε.

3. If M_b hangs on input ε, force an infinite loop.

4. If M_b halts on input ε, then run a UTM on the original input u which forces an infinite loop if u is not "M" for any TM M, and otherwise it runs M (u= "M") on input ε. If M hangs on input ε, the simulating UTM also hangs.

What language does this machine M_f accept in each of two cases:

Case 1: When machine M_b does not halt on input ε.

Case 2: When machine M_b halts on input ε.